5 OTHER TECHNICAL ISSUES

HOW TO RUN ROM CODE ON DRAM?

When we access ROM or Flash ROM, the access cycle is slow, so it makes system performance reduction. Specially, ARM core use base of boot ROM area, from 0x00 to 0x1C, as a exception vector area, so CPU should access the base address of boot ROM area when interrupt is issued. In our diagnostic code also use the base address of boot ROM, as an exception vector area. That is, when interrupt is occurred, CPU branch to exception area to fetch exception handler area.

Now, we' ll introduce the method to DRAM as a boot area. When use DRAM as boot area, the access cycle is reduced than ROM device access cycle, so the system performance is grows up.

THE PROCEDURE TO MAKE DRAM AS BOOT AREA

At the first time, ROM bank 0 used as boot ROM area. After boot, the all code on ROM bank 0 is copied to DRAM. The first all memory map is following the SNDS100 diagnostic code, ROM bank 0 base address is 0x00000000, and DRAM bank 0 base address is 0x1000000. After copy the code to DRAM, change the memory map to DRAM bank 0 as address 0x00000000, and ROM bank 0 as 0x1000000. The detail procedure to make DRAM as boot area is shown in Figure 5-1.

After doing this procedure, all boot code is running on the DRAM, even though exception vector table. The memory map difference between first boot and after change memory map is shown in Figure 5-2.

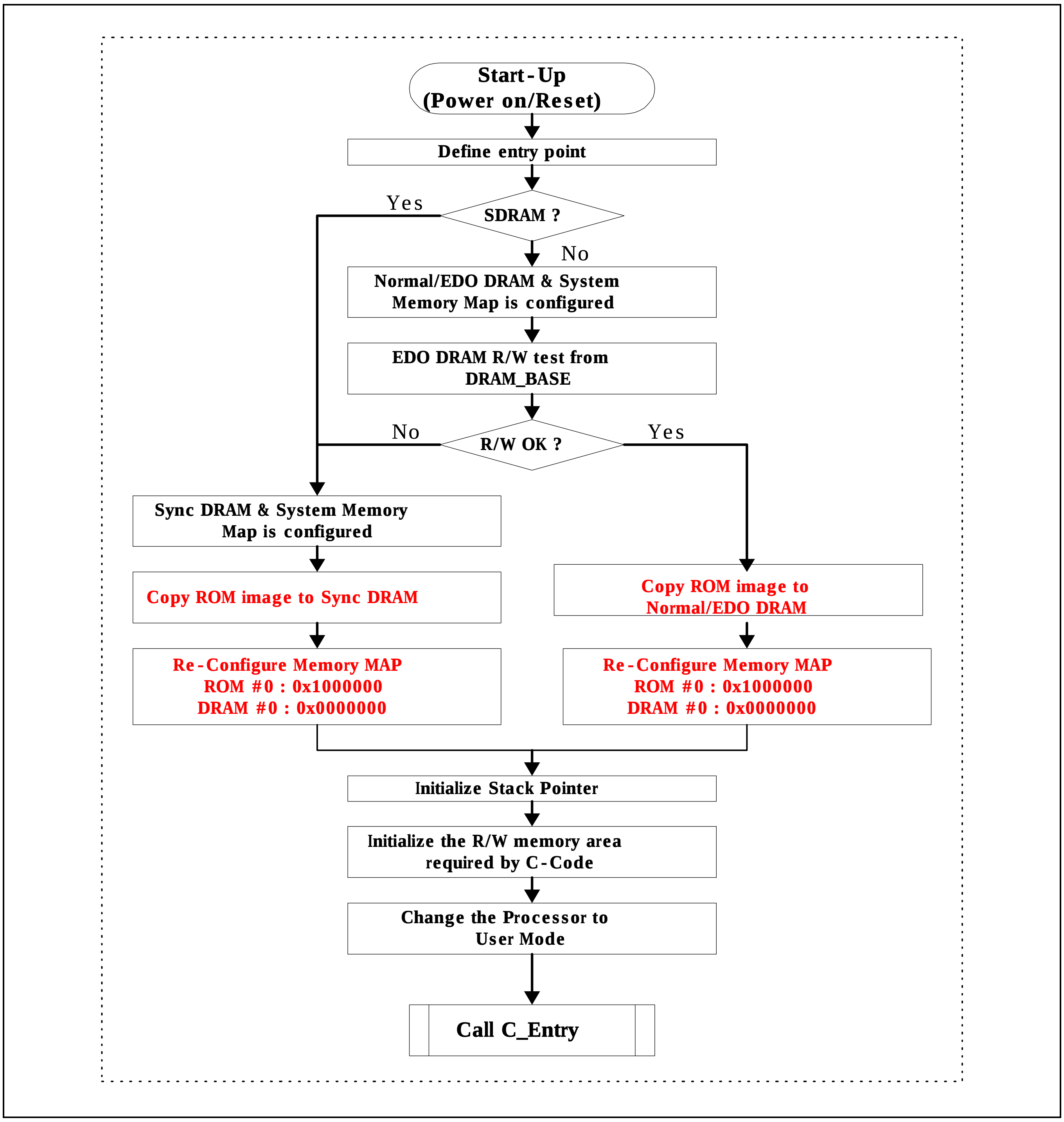

Start-Up
(Power on/Reset)
Define entry point
SDRAM ?
Yes
No
Normal/EDO DRAM & System Memory Map is configured
EDO DRAM R/W test from DRAM_BASE
R/W OK ?
No
Yes
Sync DRAM & System Memory Map is configured
Copy ROM image to Sync DRAM
Re-Configure Memory MAP ROM #0 : 0x1000000 DRAM #0 : 0x0000000
Copy ROM image to Normal/EDO DRAM
Re-Configure Memory MAP ROM #0 : 0x1000000 DRAM #0 : 0x0000000
Initialize Stack Pointer
Initialize the R/W memory area required by C-Code
Change the Processor to User Mode
Call C_Entry

Figure 5-1. The Boot Procedure When DRAM as Boot Area

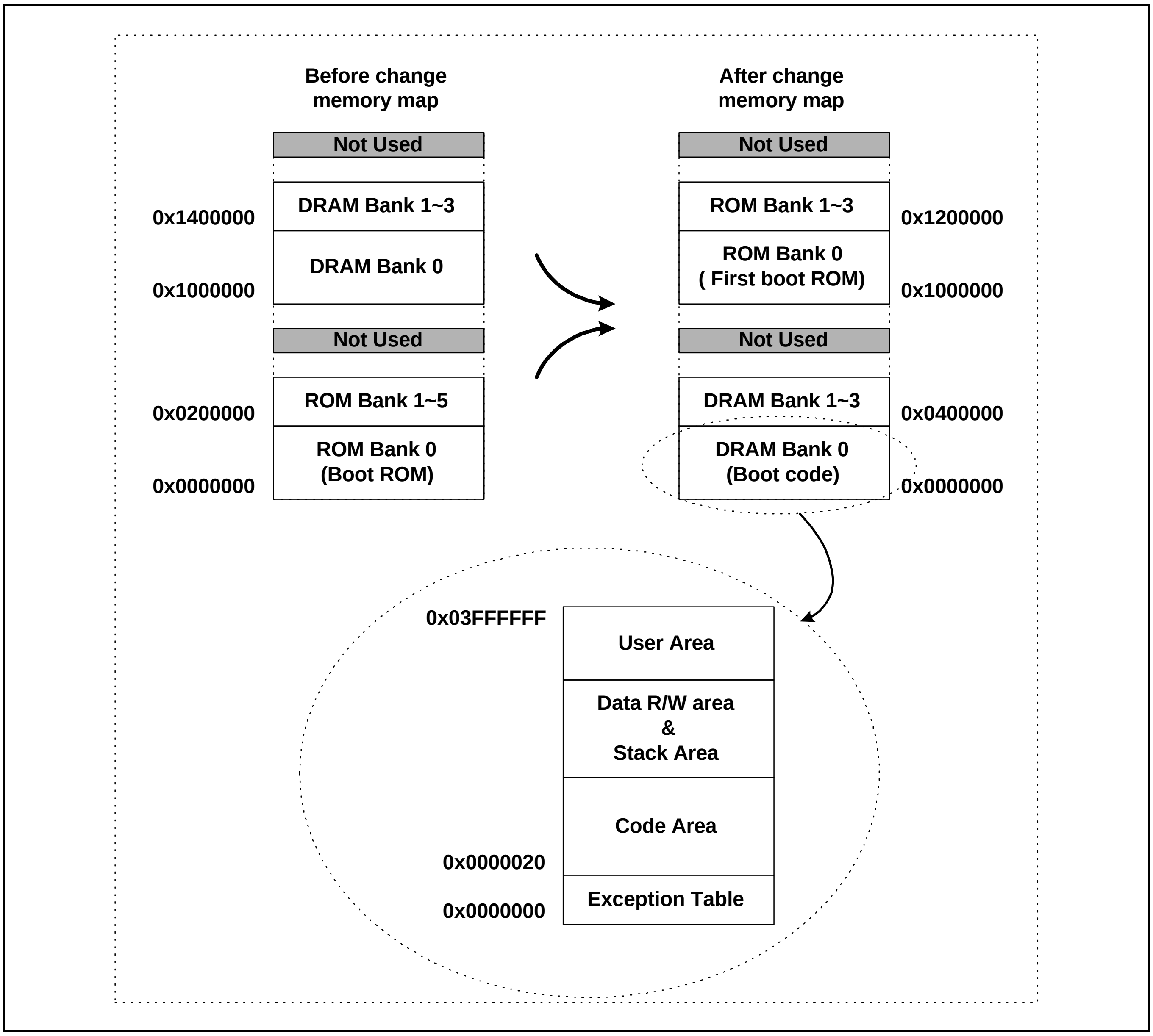


Figure 5-2. The System Memory Map When DRAM as Boot Area

THE ASSEMBLER SOURCE CODE TO MAKE DRAM AS BOOT AREA

The assembler start-up source code to make dram as boot area is some different from original SNDS100 start-up code. Some code is omitted from original SNDS100 start-up code, and some code is attached. The source code for make DRAM as boot area is described in below Listing 5-1, also in the source code, you can found the different point between original one, and the make DRAM as boot area.

Listing 5-1. Start-up Code to Make DRAM as BOOT Area (INIT.S)

```
         GET memory.a
         GET snds.a
         IMPORT  |Image$$RO$$Base|   ; Base of ROM code (=start of ROM data)
         IMPORT  |Image$$RO$$Limit|  ; End of ROM code (=start of ROM data)
         IMPORT  |Image$$RW$$Base|   ; Base of RAM to initialise
         IMPORT  |Image$$RW$$Limit|  ; Limit of RAM to initialise
         IMPORT  |Image$$ZI$$Base|   ; Base and limit of area
         IMPORT  |Image$$ZI$$Limit|  ; to zero initialise

         AREA    Init, CODE, READONLY

; --- Define entry point
     EXPORT  __main  ; defined to ensure that C runtime system
__main               ; is not linked in
     ENTRY
; --- Setup interrupt / exception vectors
   IF :DEF: ROM_AT_ADDRESS_ZERO
; If the ROM is at address 0 this is just a sequence of branches
     B     Reset_Handler
     B     SystemUndefinedHandler
     B     SystemSwiHandler
     B     SystemPrefetchHandler
     B     SystemAbortHandler
     NOP                     ; Reserved vector
     B     SystemIrqHandler
     B     SystemFiqHandler
   ELSE
; Otherwise we copy a sequence of LDR PC instructions over the vectors
; (Note: We copy LDR PC instructions because branch instructions
; could not simply be copied, the offset in the branch instruction
; would have to be modified so that it branched into ROM. Also, a
; branch instructions might not reach if the ROM is at an address
; > 32M).
     MOV    R8, #0
     ADR    R9, Vector_Init_Block
     LDMIA  R9!, {R0-R7}
     STMIA  R8!, {R0-R7}
     LDMIA  R9!, {R0-R7}
     STMIA  R8!, {R0-R7}

; Now fall into the LDR PC, Reset_Addr instruction which will continue
; execution at 'Reset_Handler'
```

```
Vector_Init_Block
       LDR     PC, Reset_Addr
       LDR     PC, Undefined_Addr
       LDR     PC, SWI_Addr
       LDR     PC, Prefetch_Addr
       LDR     PC, Abort_Addr
       NOP
       LDR     PC, IRQ_Addr
       LDR     PC, FIQ_Addr

Reset_Addr      DCD     Reset_Handler
Undefined_Addr  DCD     SystemUndefinedHandler
SWI_Addr        DCD     SystemSwiHandler
Prefetch_Addr   DCD     SystemPrefetchHandler
Abort_Addr      DCD     SystemAbortHandler
                DCD     0               ; Reserved vector
IRQ_Addr        DCD     SystemIrqHandler
FIQ_Addr        DCD     SystemFiqHandler
    ENDIF

              AREA Main, CODE, READONLY

;===========================================================
; The Reset Entry Point
;===========================================================
Reset_Handler                          ;/* Reset Entry Point */

              ;=====================================
              ; LED Display
              ;=====================================
              LDR       r1, =IOPMOD
              LDR       r0, =0xFF
              STR       r0, [r1]

              LDR       r1, =IOPDATA
              LDR       r0, =0x55
              STR       r0, [r1]

              ;=====================================
              ; Setup Special Register
              ;=====================================
              LDR       r0, =0x3FF0000       ; Read SYSCFG register value
              LDR       r1,[r0]              ; To idetify DRAM type
              LDR       r2, =0x80000000
              AND       r0, r1, r2       ; Mask DRAM type mode bit
              CMP       r0, r2
              BNE       EDO_DRAM_CONFIGURATION
              B         SYNC_DRAM_CONFIGURATION ; only when KS32C50100

              ;===================================================
              ; Special Register Configuration for EDO mode DRAM
              ; When KS32C5000 and KS32C50100
```

```
          ;================================================
EDO_DRAM_CONFIGURATION
          LDR      r0, =0x3FF0000
          LDR      r1, =0x3FFFF90     ; SetValue = 0x3FFFF91
          STR      r1, [r0]           ; Cache,WB disable
                         ; Start_addr = 0x3FF00000

          ;ROM and RAM Configuration(Multiple Load and Store)
          ADRL     r0, SystemInitData
          LDMIA    r0, {r1-r12}
          LDR      r0, =0x3FF0000 + 0x3010 ; ROMCntr Offset : 0x3010
          STMIA    r0, {r1-r12}

          LDR      r1,=DRAM_BASE
          STR      r1,[r1]     ; [DRAM_BASE] = DRAM_BASE
          LDR      r2,[r1]            ; Read DRAM Data
          CMP      r2,r1
          BNE      SYNC_DRAM_CONFIGURATION

          ;===========================================================
          ; Copy ROM image to EDO DRAM, Change ROM and DRAM Base pointer
          ;===========================================================
ROM2DRAM_COPY_START
          LDR      r0, =|Image$$RO$$Base|      ; Get pointer to ROM data
          LDR      r1, =|Image$$RW$$Limit|  ; and RAM copy
          LDR      r2, =DRAM_BASE              ; Copy DRAM area base
          SUB      r1, r1, r0          ; [r1] is loop count
          ADD      r1, r1, #4; [r1] is loop count

ROM2DRAM_COPY_LOOP
          LDR      r3, [r0], #4
          STR      r3, [r2], #4
          SUBS     r1, r1, #4; Down Count
          BNE      ROM2DRAM_COPY_LOOP

          ;==================================
          ; Change Base address of ROM and DRAM
          ;==================================
          ADRL     r0, SystemInitData_S
          LDMIA    r0, {r1-r12}
          LDR      r0, =0x3FF0000 + 0x3010 ; ROMCntr Offset : 0x3010
          STMIA    r0, {r1-r12}

          B        INITIALIZE_STACK

          ;================================================
          ; Special Register Configuration for SYNC DRAM
          ; Only when KS32C50100
          ;================================================
SYNC_DRAM_CONFIGURATION
          LDR      r0, =0x3FF0000
          LDR      r1, =0x83FFFF90    ; SetValue = 0x83FFFF91
          STR      r1, [r0]           ; Cache,WB disable
```

```
                                    ; Start_addr = 0x3FF00000

            ;ROM and RAM Configuration(Multiple Load and Store)
            ADRL    r0, SystemInitDataSDRAM
            LDMIA   r0, {r1-r12}
            LDR     r0, =0x3FF0000 + 0x3010 ; ROMCntr Offset : 0x3010
            STMIA   r0, {r1-r12}

            ;==============================================================
            ; Copy ROM image to SYNC DRAM, Change ROM and DRAM Base pointer
            ;==============================================================
ROM2SDRAM_COPY_START
            LDR     r0, =|Image$$RO$$Base|     ; Get pointer to ROM data
            LDR     r1, =|Image$$RO$$Limit|  ; and RAM copy
            LDR     r2, =DRAM_BASE             ; Copy DRAM area base
            SUB     r1, r1, r0                 ; [r1] is loop count
            ADD     r1, r1, #4              ; [r1] is loop count

ROM2SDRAM_COPY_LOOP
            LDR     r3, [r0], #4
            STR     r3, [r2], #4
            SUBS    r1, r1, #4              ; Down Count
            BNE     ROM2SDRAM_COPY_LOOP

            ;====================================
            ; Change Base address of ROM and DRAM
            ;====================================
            ADRL    r0, SystemInitDataSDRAM_S
            LDMIA   r0, {r1-r12}
            LDR     r0, =0x3FF0000 + 0x3010 ; ROMCntr Offset : 0x3010
            STMIA   r0, {r1-r12}

            ;=====================================
            ; Initialise STACK
            ;=====================================
INITIALIZE_STACK
            MRS     r0, cpsr
            BIC     r0, r0, #LOCKOUT | MODE_MASK
            ORR     r2, r0, #USR_MODE

            ORR     r1, r0, #LOCKOUT | FIQ_MODE
            MSR     cpsr, r1
            MSR     spsr, r2
            LDR     sp, =FIQ_STACK

            ORR     r1, r0, #LOCKOUT | IRQ_MODE
            MSR     cpsr, r1
            MSR     spsr, r2
            LDR     sp, =IRQ_STACK

            ORR     r1, r0, #LOCKOUT | ABT_MODE
            MSR     cpsr, r1
            MSR     spsr, r2
            LDR       sp, =ABT_STACK

            ORR       r1, r0, #LOCKOUT | UDF_MODE
            MSR       cpsr, r1
            MSR       spsr, r2
            LDR       sp, =UDF_STACK

            ORR       r1, r0, #LOCKOUT | SUP_MODE
            MSR       cpsr, r1
            MSR       spsr, r2
            LDR       sp, =SUP_STACK   ; Change CPSR to SVC mode

            ;====================================
            ; Initialise memory required by C code
            ;====================================
            LDR  r0, =|Image$$RO$$Limit| ; Get pointer to ROM data
            LDR  r1, =|Image$$RW$$Base|  ; and RAM copy
            LDR  r3, =|Image$$ZI$$Base|  ; Zero init base => top of initialised data
            CMP     r0, r1              ; Check that they are different
            BEQ     %1
0           CMP     r1, r3              ; Copy init data
            LDRCC   r2, [r0], #4
            STRCC   r2, [r1], #4
            BCC     %0
1           LDR     r1, =|Image$$ZI$$Limit| ; Top of zero init segment
            MOV     r2, #0
2           CMP     r3, r1              ; Zero init
            STRCC   r2, [r3], #4
            BCC     %2

            ;===================================================
            ; Now change to user mode and set up user mode stack.
            ;===================================================
        MRS     r0, cpsr
            BIC       r0, r0, #LOCKOUT | MODE_MASK
            ORR       r1, r0, #USR_MODE
            MSR       cpsr, r0
            LDR       sp, =USR_STACK

;       /* Call C_Entry application routine with a pointer to the first */
;       /* available memory address after ther compiler's global data   */
;       /* This memory may be used by the application.                  */
            ;==========================
            ; Now we enter the C Program
            ;==========================

        IMPORT  C_Entry
        BL      C_Entry

;==========================================
; Exception Vector Function Definition
; Consist of function Call from C-Program.
;==========================================
SystemUndefinedHandler
        IMPORT ISR_UndefHandler
        STMFD   sp!, {r0-r12}
        B       ISR_UndefHandler
        LDMFD   sp!, {r0-r12, pc}^

SystemSwiHandler
        STMFD   sp!, {r0-r12,lr}
        LDR     r0, [lr, #-4]
        BIC     r0, r0, #0xff000000
        CMP     r0, #0xff
        BEQ     MakeSVC
        LDMFD   sp!, {r0-r12, pc}^
MakeSVC
        MRS     r1, spsr
        BIC     r1, r1, #MODE_MASK
        ORR     r2, r1, #SUP_MODE
        MSR     spsr, r2
        LDMFD   sp!, {r0-r12, pc}^

SystemPrefetchHandler
        IMPORT ISR_PrefetchHandler
        STMFD   sp!, {r0-r12, lr}
        B       ISR_PrefetchHandler
        LDMFD   sp!, {r0-r12, lr}
        ;ADD    sp, sp, #4
        SUBS    pc, lr, #4

SystemAbortHandler
        IMPORT ISR_AbortHandler
        STMFD   sp!, {r0-r12, lr}
        B       ISR_AbortHandler
        LDMFD   sp!, {r0-r12, lr}
        ;ADD    sp, sp, #4
        SUBS    pc, lr, #8

SystemReserv
        SUBS    pc, lr, #4

SystemIrqHandler
        IMPORT ISR_IrqHandler
        STMFD   sp!, {r0-r12, lr}
        BL      ISR_IrqHandler
        LDMFD   sp!, {r0-r12, lr}
        SUBS    pc, lr, #4

SystemFiqHandler
        IMPORT ISR_FiqHandler
        STMFD   sp!, {r0-r7, lr}
        BL      ISR_FiqHandler
        LDMFD   sp!, {r0-r7, lr}
        SUBS    pc, lr, #4
```

```
            AREA ROMDATA, DATA, READONLY

;=====================================================
; DRAM System Initialize Data(KS32C5000 and KS32C50100)
;=====================================================
SystemInitData
            DCD rEXTDBWTH           ; DRAM1(Half), ROM5(Byte), ROM1(Half), else 32bit
            DCD rROMCON0        ; 0x0000000 ~ 0x01FFFFF, ROM0,4Mbit,2cycle
            DCD rROMCON1        ;
            DCD rROMCON2        ; 0x0400000 ~ 0x05FFFFF, ROM2
            DCD rROMCON3        ; 0x0600000 ~ 0x07FFFFF, ROM3
            DCD rROMCON4        ; 0x0800000 ~ 0x09FFFFF, ROM4
            DCD rROMCON5        ;
            DCD rDRAMCON0       ; 0x1000000 ~ 0x13FFFFF, DRAM0 4M,
            DCD rDRAMCON1           ; 0x1400000 ~ 0x17FFFFF, DRAM1 4M,
            DCD rDRAMCON2           ; 0x1800000 ~ 0x1EFFFFF, DRAM2 16M
            DCD rDRAMCON3           ; 0x1C00000 ~ 0x1FFFFFF
            DCD rREFEXTCON      ; External I/O, Refresh

SystemInitData_S
            DCD rEXTDBWTH           ; DRAM1(Half), ROM5(Byte), ROM1(Half), else 32bit
            DCD rROMCON0_S          ; 0x1000000 ~ 0x11FFFFF, ROM0,4Mbit,2cycle
            DCD rROMCON1_S          ; 0x1200000 ~ 0x13FFFFF, ROM0
            DCD rROMCON2_S          ; 0x1400000 ~ 0x15FFFFF, ROM2
            DCD rROMCON3_S          ; 0x1600000 ~ 0x17FFFFF, ROM3
            DCD rROMCON4_S          ; 0x1800000 ~ 0x19FFFFF, ROM4
            DCD rROMCON5_S          ; 0x1A00000 ~ 0x1BFFFFF, ROM4
            DCD rDRAMCON0_S     ; 0x0000000 ~ 0x03FFFFF, DRAM0
            DCD rDRAMCON1_S         ; 0x0400000 ~ 0x07FFFFF, DRAM1
            DCD rDRAMCON2_S         ; 0x0800000 ~ 0x0EFFFFF, DRAM2
            DCD rDRAMCON3_S         ; 0x0C00000 ~ 0x0FFFFFF
            DCD rREFEXTCON      ; External I/O, Refresh

;=====================================================
; SDRAM System Initialize Data          (KS32C50100 only)
;=====================================================
SystemInitDataSDRAM
            DCD rEXTDBWTH           ; DRAM1(Half), ROM5(Byte), ROM1(Half), else 32bit
            DCD rROMCON0        ; 0x0000000 ~ 0x01FFFFF, ROM0,4Mbit,2cycle
            DCD rROMCON1        ;
            DCD rROMCON2        ; 0x0400000 ~ 0x05FFFFF, ROM2
            DCD rROMCON3        ; 0x0600000 ~ 0x07FFFFF, ROM3
            DCD rROMCON4        ; 0x0800000 ~ 0x09FFFFF, ROM4
            DCD rROMCON5        ;
            DCD rSDRAMCON0      ; 0x1000000 ~ 0x13FFFFF, DRAM0 4M,
            DCD rSDRAMCON1          ; 0x1400000 ~ 0x17FFFFF, DRAM1 4M,
            DCD rSDRAMCON2          ; 0x1800000 ~ 0x1EFFFFF, DRAM2 16M
            DCD rSDRAMCON3          ; 0x1C00000 ~ 0x1FFFFFF
            DCD rSREFEXTCON     ; External I/O, Refresh

SystemInitDataSDRAM_S
            DCD rEXTDBWTH           ; DRAM1(Half), ROM5(Byte), ROM1(Half), else 32bit
            DCD rROMCON0_S           ; 0x1000000 ~ 0x11FFFFF, ROM0,4Mbit,2cycle
            DCD rROMCON1_S           ;
            DCD rROMCON2_S           ; 0x1400000 ~ 0x15FFFFF, ROM2
            DCD rROMCON3_S           ; 0x1600000 ~ 0x17FFFFF, ROM3
            DCD rROMCON4_S           ; 0x1800000 ~ 0x19FFFFF, ROM4
            DCD rROMCON5_S           ;
            DCD rSDRAMCON0_S ; 0x0000000 ~ 0x03FFFFF, DRAM0 4M,
            DCD rSDRAMCON1_S ; 0x0400000 ~ 0x07FFFFF, DRAM1 4M,
            DCD rSDRAMCON2_S ; 0x0800000 ~ 0x0EFFFFF, DRAM2 16M
            DCD rSDRAMCON3_S ; 0x0C00000 ~ 0x0FFFFFF
            DCD rSREFEXTCON  ; External I/O, Refresh

            ALIGN

;/*****************************************************/
            AREA SYS_STACK, NOINIT
;/*****************************************************/
            %       USR_STACK_SIZE
USR_STACK
            %       UDF_STACK_SIZE
UDF_STACK
            %       ABT_STACK_SIZE
ABT_STACK
            %       IRQ_STACK_SIZE
IRQ_STACK
            %       FIQ_STACK_SIZE
FIQ_STACK
            %       SUP_STACK_SIZE
SUP_STACK

;/*****************************************************/

            END
```

When use DRAM as boot area, then you also need to modify the "snds.a" file. This file define the memory controller base and end pointer, the source code of "snds.a" is listed Listing 5-2.

Listing 5-2. Memory Control Value Definition (SNDS.A)

```
;-------------------------------------------------------------
SOME CODE OMITTED FOR READIBILITY

;/* -> ROMCON0 : ROM Bank0 Control register */
;-------------------------------------------------------------
ROMBasePtr0     EQU  0x000:SHL:10    ;=0x0000000 ; normal mode
ROMEndPtr0      EQU  0x020:SHL:20    ;=0x0200000 ; normal mode
ROMBasePtr0_S   EQU  0x100:SHL:10    ;=0x1000000
ROMEndPtr0_S    EQU  0x120:SHL:20    ;=0x1200000

PMC0            EQU  0x0             ; 0x0=Normal ROM, 0x1=4Word Page
                                     ; 0x2=8Word Page, 0x3=16Word Page
rTpa0           EQU  (0x0:SHL:2)     ; 0x0=5Cycle, 0x1=2Cycle
```

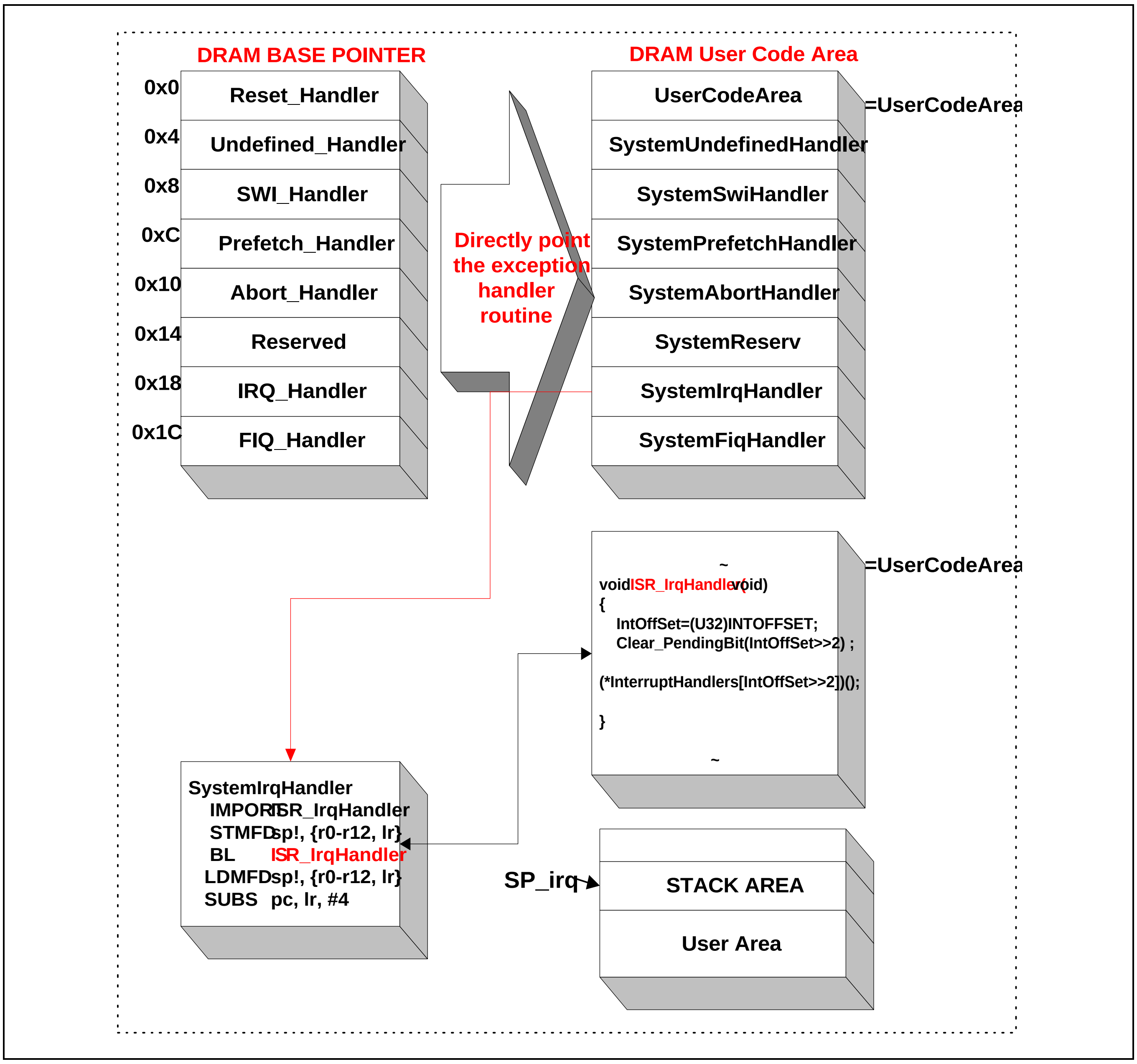
```
                                    ; 0x2=3Cycle, 0x3=4Cycle
rTacc0         EQU  (0x6:SHL:4)    ; 0x0=Disable, 0x1=2Cycle
                                    ; 0x2=3Cycle, 0x3=4Cycle
                                    ; 0x4=5Cycle, 0x5=6Cycle
                                    ; 0x6=7Cycle, 0x7=Reserved

rROMCON0      EQU  ROMEndPtr0+ROMBasePtr0+rTacc0+rTpa0+PMC0
rROMCON0_S             EQU  ROMEndPtr0_S+ROMBasePtr0_S+rTacc0+rTpa0+PMC0
;------------------------------------------------------------
```

SOME CODE OMITTED FOR READIBILITY

```
;/* -> DRAMCON0 : RAM Bank0 control register */
;------------------------------------------------------------
EDO_Mode0         EQU  1                    ;(EDO)0=Normal, 1=EDO DRAM
CasPrechargeTime0  EQU  1                    ;(Tcp)0=1cycle,1=2cycle
CasStrobeTime0     EQU  3                   ;(Tcs)0=1cycle ~ 3=4cycle
DRAMCON0Reserved   EQU  1                    ; Must be set to 1

RAS2CASDelay0     EQU  1                    ;(Trc)0=1cycle,1=2cycle
RASPrechargeTime0  EQU  3                    ;(Trp)0=1cycle ~ 3=4clcyle
DRAMBasePtr0       EQU  0x100:SHL:10          ;=0x1000000
DRAMEndPtr0       EQU  0x140:SHL:20          ;=0x1400000
DRAMBasePtr0_S     EQU  0x000:SHL:10          ;=0x0000000
DRAMEndPtr0_S     EQU  0x040:SHL:20          ;=0x0400000
NoColumnAddr0      EQU  2                   ;0=8bit,1=9bit,2=10bit,3=11bits
;------------------------------------------------------------
Tcs0              EQU  CasStrobeTime0:SHL:1
Tcp0              EQU  CasPrechargeTime0:SHL:3
dumy0              EQU  DRAMCON0Reserved:SHL:4 ; dummy cycle

Trc0              EQU  RAS2CASDelay0:SHL:7
Trp0              EQU  RASPrechargeTime0:SHL:8
CAN0              EQU  NoColumnAddr0:SHL:30

rDRAMCON0    EQU  CAN0+DRAMEndPtr0+DRAMBasePtr0+Trp0+Trc0+Tcp0+Tcs0+dumy0+EDO_Mode0
rDRAMCON0_S           EQU
CAN0+DRAMEndPtr0_S+DRAMBasePtr0_S+Trp0+Trc0+Tcp0+Tcs0+dumy0+EDO_Mode0
;--------------------------------------------------------------------------------

SRAS2CASDelay0     EQU  1                    ;(Trc)0=1cycle,1=2cycle
SRASPrechargeTime0 EQU  3                    ;(Trp)0=1cycle ~ 3=4clcyle
SNoColumnAddr0     EQU  0                   ;0=8bit,1=9bit,2=10bit,3=11bits
SCAN0             EQU  SNoColumnAddr0:SHL:30
STrc0             EQU  SRAS2CASDelay0:SHL:7
STrp0             EQU  SRASPrechargeTime0:SHL:8
;
rSDRAMCON0     EQU   SCAN0+DRAMEndPtr0+DRAMBasePtr0+STrp0+STrc0
rSDRAMCON0_S      EQU SCAN0+DRAMEndPtr0_S+DRAMBasePtr0_S+STrp0+STrc0
;------------------------------------------------------------
```

EXCEPTION HANDLING WHEN USE DRAM AS BOOT AREA

When use DRAM as boot area, the exception handle routine is easy to implement. At the original diagnostic code use DRAM base pointer as exception handler area, but when you use DRAM as boot area, you only need the exception vector area to going into interrupt service routine. The exception handle routine is depicted in Figure 5-3. This figure show the exception handling routine example with IRQ exception.



```
~
void ISR_IrqHandler(void)
{
   IntOffSet=(U32)INTOFFSET;
   Clear_PendingBit(IntOffSet>>2) ;

(*InterruptHandlers[IntOffSet>>2])();

}

                    ~
```

```
SystemIrqHandler
   IMPORT ISR_IrqHandler
   STMFD sp!, {r0-r12, lr}
   BL     ISR_IrqHandler
  LDMFD sp!, {r0-r12, lr}
  SUBS  pc, lr, #4
```

Figure 5-3. Exception Handling Process When Use DRAM as Boot Area

HOW TO DEBUG SNDS100 USING EMULATOR WITHOUT BOOT ROM?

You can evaluate SNDS100 board using circuit emulator(ex, EmbeddedICE) without boot rom. But, to do this, you have to consider two points. One is the side of hardware consideration and the other is software consideration.

That is due to the reset circuit(nRESET, nTRST) problem of NetMCU device. SNDS100 reset circuit have to be revised which is refered to “Section 4. System Design : Figure 4-28 EmbededICE interface design example”.

This is came from the without boot ROM. After the power on reset or key reset, ROM bank0 will be located at address zero by default reset system memory map but there is no boot ROM. So, you have to re-map the system default memory map before downloading the application. This memory map can be changed by call initial map file at ADW’ s command window.(Arm Debuger window).

HARDWARE CONSIDERATION

It is the main problem of invoking ADW (ARM Debugger) after resetting SNDS100 board that SWI interrupts the downloading of a debugging program . We find out that nTRST(62-th pin : pull-up internally) is needed to be floated. Just cutting the pattern connected to nTRST pin in SNDS100 board , you can invoke ADW and download a program without any problem and without considering time interval. (See, Figure 4-28).

SOFTWARE CONSIDERATION

After modifying SNDS100 board , you can go to the next step as following flow.

— Making a debugging image file

— Invoking ADW

— Setting the special registers of KS32C5000 series and ADW for debugging.

— Downloading Image file.

— Step into Debugging mode

The downloaded program will start and you can debug your program with the ADW.

Making a debugging image file

You need to modify files concerning memory configuration and interrupt handling.

These are init.s, memory.a and makefile

Modifying memory.a and snds.a

Some value related to memory configuration should be changed in a pre-distributed memory. a file.

— DRAM_Base symbol value

— (This means downloading start area, it is distinguished form booting code we tried)

```
DRAM_Base  EQU 0x1000000          (old)
DRAM_Base  EQU 0x300000           (new)
```

— System User Area

```
^           DRAM_Base+ExceptionSize ;
; ^         0x1000050                    (old )
; ^         0x300050 ;                   (new)
```

Only one symbol is needed to be changed for snds.a

```
DRAM_BASE  EQU 0x1000000          (old)
DRAM_BASE  EQU 0x300000           (new)
```

Modifying init.s

Inis.s file is simplified only for downloading DRAM. The main difference from pre-distributed one is that Memory Initialized (SystemInitData symbol) part is removed. Part 2.3 Setting Special Register for KS32C50100 will explain how to replace this role . And we overwrite IRQ_handler address at 0x18 address and FIQ_handler address at 0x1c address for interrupt mapping . This may corrupt a debugging information , but it will is correspond to only debug starting information. So you can keep debugging with ADW without regarding this.

Listing 5-3. Modifying Init.s

```
  GET memory.a
  GET snds.a

  AREA    Init, CODE, READONLY

; --- Define entry point
      EXPORT  __main  ; defined to ensure that C runtime system
__main              ; is not linked in
      ENTRY
          B      Reset_Handler
      B      Undefined_Handler
      B      SWI_Handler
      B      Prefetch_Handler
      B      Abort_Handler
      NOP                        ; Reserved vector
      B      IRQ_Handler
      B      FIQ_Handler
```

```
;==============================================================
; The Default Exception Handler Vector Entry Pointer Setup
;==============================================================
  nop
FIQ_Handler
  SUB      sp, sp, #4
  STMFD  sp!, {r0}
  LDR      r0, =HandleFiq
  LDR      r0, [r0]
  STR      r0, [sp, #4]
  LDMFD  sp!, {r0, pc}

IRQ_Handler
  SUB      sp, sp, #4
  STMFD  sp!, {r0}
  LDR      r0, =HandleIrq
  LDR      r0, [r0]
  STR      r0, [sp, #4]
  LDMFD  sp!, {r0, pc}

Prefetch_Handler
  SUB      sp, sp, #4
  STMFD  sp!, {r0}
  LDR      r0, =HandlePrefetch
  LDR      r0, [r0]
  STR      r0, [sp, #4]
  LDMFD  sp!, {r0, pc}

Abort_Handler
  SUB      sp, sp, #4
  STMFD  sp!, {r0}
  LDR      r0, =HandleAbort
  LDR      r0, [r0]
  STR      r0, [sp, #4]
  LDMFD  sp!, {r0, pc}

Undefined_Handler
  SUB      sp, sp, #4
  STMFD  sp!, {r0}
  LDR      r0, =HandleUndef
  LDR      r0, [r0]
  STR      r0, [sp, #4]
  LDMFD  sp!, {r0, pc}

SWI_Handler
  SUB      sp, sp, #4
  STMFD  sp!, {r0}
  LDR      r0, =HandleSwi
  LDR      r0, [r0]
  STR      r0, [sp, #4]
  LDMFD  sp!, {r0, pc}


  AREA Main, CODE, READONLY

;====================================
; The Reset Entry Point
;====================================
Reset_Handler
```

```
;=====================================
; Initialise STACK
;=====================================
INITIALIZE_STACK
  MRS      r0, cpsr
  BIC      r0, r0, #LOCKOUT | MODE_MASK
  ORR      r2, r0, #USR_MODE

  ORR      r1, r0, #LOCKOUT | FIQ_MODE
  MSR      cpsr, r1
  MSR      spsr, r2
  LDR      sp, =FIQ_STACK

  ORR      r1, r0, #LOCKOUT | IRQ_MODE
  MSR      cpsr, r1
  MSR      spsr, r2
  LDR      sp, =IRQ_STACK

  ORR      r1, r0, #LOCKOUT | ABT_MODE
  MSR      cpsr, r1
  MSR      spsr, r2
  LDR      sp, =ABT_STACK

  ORR      r1, r0, #LOCKOUT | UDF_MODE
  MSR      cpsr, r1
  MSR      spsr, r2
  LDR      sp, =UDF_STACK

  ORR      r1, r0, #LOCKOUT | SUP_MODE
  MSR      cpsr, r1
  MSR      spsr, r2
  LDR      sp, =SUP_STACK   ; Change CPSR to SVC mode


;=====================================
; LED Display
;=====================================
  LDR      r1, =IOPMOD
  LDR      r0, =0xFF
  STR      r0, [r1]

  LDR      r1, =IOPDATA
  LDR      r0, =0x55
  STR      r0, [r1]

;=====================================
; UART Setup for Console
; 38400, data= 8 bits , stop = 1 bits
;=====================================
  LDR      r1, =UARTLCON0
  LDR      r0, =0x43
  STR      r0,[r1]

  LDR      r1, =UARTCONT0
  LDR      r0, =0x9
  STR      r0,[r1]

  LDR      r1, =UARTBRD0
; LDR      r0, =0x280 ; 38400 bps for 50MHz
; LDR      r0, =0x350 ; 38400 bps for 33MHz
  LDR      r0, =0x2f0 ; 38400 bps for 29.4912MHz
; ;LDR     r0, =0x160 ; 38400 bps for 14.31818MHz
  STR      r0,[r1]
  ; Print out   Boot Up Banner  to Console
  LDR       r0, =BootUpBanner0
  BL        PrintString

;  Main Memory Test (POST)
  LDR       r0, =DramPostStart
  BL        PrintString

;==================================================
; Special Register Configuration for EDO mode DRAM
;==================================================
  B         EXCEPTION_VECTOR_TABLE_SETUP

;==================================================
; Special Register Configuration for SYNC DRAM
;==================================================

;=============================
; Exception Vector Table Setup
;=============================
EXCEPTION_VECTOR_TABLE_SETUP
  LDR       r0, =HandleReset; Exception Vector Table Memory Loc.
  LDR       r1, =ExceptionHandlerTable ; Exception Handler Assign
  MOV       r2, #8                     ; Number of Exception is 8
ExceptLoop
  LDR       r3, [r1], #4
  STR       r3, [r0], #4
  SUBS      r2, r2, #1                 ; Down Count
  BNE       ExceptLoop

;=====================================
; Initialise memory required by C code
;=====================================
            IMPORT  |Image$$RO$$Limit|  ; End of ROM code (=start of ROM data)
            IMPORT  |Image$$RW$$Base|   ; Base of RAM to initialise
            IMPORT  |Image$$ZI$$Base|   ; Base and limit of area
            IMPORT  |Image$$ZI$$Limit|  ; to zero initialise

            LDR  r0, =|Image$$RO$$Limit| ; Get pointer to ROM data
            LDR  r1, =|Image$$RW$$Base|  ; and RAM copy
            LDR  r3, =|Image$$ZI$$Base|  ; Zero init base => top of initialised data
            CMP     r0, r1               ; Check that they are different
            BEQ     %1
0           CMP     r1, r3               ; Copy init data
            LDRCC   r2, [r0], #4
            STRCC   r2, [r1], #4
            BCC     %0
1           LDR     r1, =|Image$$ZI$$Limit| ; Top of zero init segment
            MOV     r2, #0
2           CMP     r3, r1               ; Zero init
            STRCC   r2, [r3], #4
            BCC     %2

;====================================================
; Now change to user mode and set up user mode stack.
;====================================================
     MRS     r0, cpsr
  BIC       r0, r0, #LOCKOUT | MODE_MASK
  ORR       r1, r0, #USR_MODE
  MSR       cpsr, r0
  LDR       sp, =USR_STACK

;     /* Call C_Entry application routine with a pointer to the first */
```

SAMSUNG ELECTRONICS

```
;       /* available memory address after ther compiler's global data   */
;       /* This memory may be used by the application.                  */
;===========================
; Now we enter the C Program
;===========================

  LDR       r0, =0x98
  LDR       r1, =0x18
  LDR       r2, [r0]
  STR       r2, [r1]

;  This is for IRQ_handler

  LDR       r0, =0x9c
  LDR       r1, =0x1c
  LDR       r2, [r0]
  STR       r2, [r1]

;  This is for FIQ_handler

   IMPORT  C_Entry
   BL      C_Entry

;==========================================
; Exception Vector Function Definition
; Consist of function Call from C-Program.
;==========================================
SystemUndefinedHandler
  IMPORT  ISR_UndefHandler
  STMFD   sp!, {r0-r12}
  B       ISR_UndefHandler
  LDMFD   sp!, {r0-r12, pc}^

SystemSwiHandler
  STMFD   sp!, {r0-r12,lr}
  LDR     r0, [lr, #-4]
  BIC     r0, r0, #0xff000000
  CMP     r0, #0xff
  BEQ     MakeSVC
  LDMFD   sp!, {r0-r12, pc}^
MakeSVC
  MRS     r1, spsr
  BIC     r1, r1, #MODE_MASK
  ORR     r2, r1, #SUP_MODE
  MSR     spsr, r2
  LDMFD   sp!, {r0-r12, pc}^

SystemPrefetchHandler
  IMPORT  ISR_PrefetchHandler
  STMFD   sp!, {r0-r12, lr}
  B       ISR_PrefetchHandler
  LDMFD   sp!, {r0-r12, lr}
  ;ADD    sp, sp, #4
  SUBS    pc, lr, #4

SystemAbortHandler
  IMPORT  ISR_AbortHandler
  STMFD   sp!, {r0-r12, lr}
  B       ISR_AbortHandler
  LDMFD   sp!, {r0-r12, lr}
  ;ADD    sp, sp, #4
  SUBS    pc, lr, #8

SystemReserv
```

```
  SUBS     pc, lr, #4

SystemIrqHandler
  IMPORT ISR_IrqHandler
  STMFD  sp!, {r0-r12, lr}
  BL         ISR_IrqHandler
  LDMFD  sp!, {r0-r12, lr}
  SUBS     pc, lr, #4

SystemFiqHandler
  IMPORT ISR_FiqHandler
  STMFD  sp!, {r0-r7, lr}
  BL         ISR_FiqHandler
  LDMFD  sp!, {r0-r7, lr}
  SUBS     pc, lr, #4

;====================================================
;                          Utility Section
;               Print a zero terminated string
;====================================================
PrintString
  MOV                   r4, lr
  MOV                   r5, r0
01          LDRB                 r1, [r5],#1                ; read Byte( one character)
  AND       r0,r1, #&ff
  TST                   r0, #&ff
  MOVEQ                 pc, r4                  ; if 0, return.
  BL                    PutByte
  B                     %B01

PutByte
PutByteLoop
  LDR       r3,=UARTSTAT0
  LDR       r2,[r3]
  TST       r2,#&40                   ; UART_STAT_XMIT_EMPTY ?
  BEQ       PutByteLoop
  LDR       r3,=UARTTXH0
  STR       r0,[r3]                 ; write to THR
  MOV      pc,lr


  AREA ROMDATA, DATA, READONLY

;==========================================
; Exception Handler Vector Table Entry Point
;==========================================
ExceptionHandlerTable
  DCD       UserCodeArea
  DCD       SystemUndefinedHandler
  DCD       SystemSwiHandler
  DCD       SystemPrefetchHandler
  DCD       SystemAbortHandler
  DCD       SystemReserv
  DCD       SystemIrqHandler
  DCD       SystemFiqHandler

  ALIGN

BootUpBanner0     DCB      &a,&d,&a,&d,"$$$   SNDS100 Boot-Up Dialog !!!",0
DramPostStart     DCB      &a,&d,"$$$   SNDS100 DRAM Post => ",0
DramPostFinish    DCB      "MB Installed !",0
CacheEnableBanner           DCB      &a,&d," $$$   8K Cache is Enabled !",0

;/**************************************************/
```

```
  AREA SYS_STACK, NOINIT
;/****************************************************/
              %      USR_STACK_SIZE
USR_STACK
              %      UDF_STACK_SIZE
UDF_STACK
              %      ABT_STACK_SIZE
ABT_STACK
              %      IRQ_STACK_SIZE
IRQ_STACK
              %      FIQ_STACK_SIZE
FIQ_STACK
              %      SUP_STACK_SIZE
SUP_STACK

;/****************************************************/

  END
```

Modifying makefile

You should map Code area to address 0x0 and R/W area to a reasonable address below DRAM_Base. The following list shows the way to modify a makefile .

Listing 5-4. Makefile

```
TOOLPath          = C:\arm211a\bin
LINK      = $(TOOLPath)\armlink
ASM       = $(TOOLPath)\armasm
CC        = $(TOOLPath)\armcc
LIB      = C:\arm211a\lib\armlib.32b

ASMFLAG1          = -bi -g -apcs 3/32bit
CFLAGS= -bi -c -fc -g -apcs 3/32bit -processor ARM7TM -arch 4T

## >> When code down load to DRAM
ASMFLAG3          = -bi -apcs 3/32bit -g -PD "ROM_AT_ADDRESS_ZERO SETL {TRUE}"
LINKFLAG4         = -first init.o(Init) -RO 0x0 -RW 0x200000 -o diag.axf -aif -debug -Symbols diag.sym
$(OBJS) $(LIB)
OBJS          =  init.o start.o diag.o yours.o

diag.axf: $(OBJS)
     $(LINK) $(LINKFLAG4)
init.o: init.s
       $(ASM) $(ASMFLAG3)  init.s -o init.o -list init.lst
start.o: start.s
$(ASM) $(ASMFLAG1)  start.s -o start.o
diag.o: diag.c
$(CC) $(CFLAGS) -errors diag.err  diag.c
yours.o: yours.c
$(CC) $(CFLAGS) -errors yours.err  yours.c
```

After modifying makefile , type C:\> armmake diag.axf to get debugging image file.

Invoking ADW

If you modify SNDS100 board , ADW can be invoked without considering the elapsed time.

Setting the Special Registers of KS32C5000 series

Your write a text file to write special registers.

Your write a text file to write special registers.

When using EDO or Fast Page DRAM (armdram.ini at c root directory)

```
let 0x3ff3010 = 0xfffffffa
let 0x3ff3014 = 0x12040060
let 0x3ff3018 = 0x14042060
let 0x3ff301c = 0x16044060
let 0x3ff3020 = 0x18046060
let 0x3ff3024 = 0x1a048060
let 0x3ff3028 = 0x1c04a060
let 0x3ff302c = 0x8400039b
let 0x3ff3030 = 0x0601039b
let 0x3ff3034 = 0x0801439b
let 0x3ff3038 = 0x0a01839b
let 0x3ff303c = 0xce378360
```

When using Sync DRAM (armsdram.ini at c root directory)

```
let 0x3ff0000 = 0x87ffff91
let 0x3ff3010 = 0xfffffffa
let 0x3ff3014 = 0x12040060
let 0x3ff3018 = 0x14042060
let 0x3ff301c = 0x16044060
let 0x3ff3020 = 0x18046060
let 0x3ff3024 = 0x1a048060
let 0x3ff3028 = 0x1c04a060
let 0x3ff302c = 0x04000380
let 0x3ff3030 = 0x0601039b
let 0x3ff3034 = 0x0801439b
let 0x3ff3038 = 0x0a01839b
let 0x3ff303c = 0xce338360
```

After making text file , give a command (obey c:\armsdram.ini) at ADW command window. Then ADW initial KS32C5000 series system manager registers for interfacing with SDRAM. After this step , you have finished the preparation to download a image.

Selecting image file and Setting ADW for Debugging

Select File menu bar and Load Image menu and get your image file.Then type obey c:\armsd.ini (see Programmer' s Guide Section 2.).

Step into Debugging.

After doing all of previous work , you can run into debugging with code at DRAM based 0x0.